Modeling and Simulation of Reaction-Diffusion Problem Applied to Biofilm Growth

by

Jessica S. Armstrong

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mathematics
Honors Associate

Presented December 1st, 2014
Commencement June 2015

AN ABSTRACT OF THE THESIS OF

Jessica S. Armstrong for the degree of Honors Baccalaureate of Science in Mathematics presented on December 1st, 2014. Title: Modeling and Simulation of Reaction-Diffusion Problem Applied to Biofilm Growth.

Abstract Approved: _____
Malgorzata Peszynska

One method for CO2 sequestration is to store the liquid form of CO2 in small cement wells in the ground. An issue with this method is the susceptibility of cement to fracture and thus release the CO2. One proposition is to fill these cracks with biofilm of the bacteria Sporosarcina pasteurii (s.p.).
The goal of this project is to optimize the feeding and growth of these biofilm colonies. We will model two components of this system: chemical reactions (between food nutrients and environment) and diffusion (of nutrients).
In order to optimize this system we can model it using differential equations and numerical methods. The differential equations describe the laws which govern the situation, while the numerical methods allow us to use these equations to simulate real world data.

Key Words: Modeling, Simulation, Biofilm, Numerical Method, Diffusion, Reaction-Diffusion, Mass Reaction.

Corresponding email: jsarmstrong331@gmail.com

Modeling and Simulation of Reaction-Diffusion Problem Applied to Biofilm Growth

by

Jessica S. Armstrong

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mathematics
Honors Associate

Presented December 1st, 2014
Commencement June 2015

Honors Baccalaureate of Science in Mathematics project of Jessica S. Armstrong presented on December 1st, 2014.

APPROVED:

Malgorzata Peszynska Ph.D., Mentor, representing Mathematics

Robert L. Higdon Ph.D., Committee Member, representing Mathematics

---

Mina E. Ossiander Ph.D, Committee Member, representing Mathematics

---

Toni Doolen, Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Jessica S. Armstrong, Author

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGEMENTS

I would like to thank all of the people who helped and supported me with writing this research project.

Firstly, I would like to thank Professor Malgorzata Peszynska for guiding me through the writing and creation of this project.

Secondly, I would like to thank Connie Shen, Brittany Allen, Maria Hanoa, Paul Hanoa, and Matthew Kirsch for the support they have given me throughout my time at Oregon State University.

Thirdly, I would like to recognize Connie Shen, Peter Kagey, Professor Malgorzata Peszynska, and Professor Robert L. Higdon for proof reading and helping me understand crucial concepts for this paper.

Lastly, I would like to thank Professor Mina Ossiander, Professor Robert L. Higdon, Professor Malgorzata Peszynska, the OSU University Honors College, the OSU Mathematics Department, the College of Science Dean, Sastry Pantula, my academic advisor Professor, William A. Bogley, and the Math Chair Professor, Thomas P. Dick, for their academic and personal assistance to myself and this project.

1. Introduction

As atmospheric CO_2 continues to be the primary greenhouse gas contributing to global warming, CO_2 sequestration is one of the leading efforts to reduce the effects of climate change. The capture and storage of atmospheric CO_2 in cement wells has been shown to be a viable and long-lasting method for sequestration [1]. Research article [1] describes the injection of supercritical CO_2, a fluid state of carbon dioxide, into man-made, underground wells where it dissolves in brine and binds with rock formations for long-term storage [1].

One issue with this method of sequestration is that ground movement creates tiny fractures in the cement through which carbon dioxide leaks. Additionally, cap-rock is permanently characterized by a significant level of permeability [1]. Researchers seek to use microbial biofilms, bacterial colonies that adhere to surfaces using slimy excretions, to plug these cement cracks. In particular, the bacteria Sporosarcina pastuerii is being studied for use because of its ability to extract calcium carbonate from its surrounding environment. Sporosarcina pastuerii then produces calcite, a hardened substance, in a process called biocementation. This biofilm and calcite fills the cement cracks and inhibits upward leakage of CO_2 [1]. Researchers seek to control and encourage the growth of Sporosarcina pastuerii to make underground sequestraion effective.

This paper is motivated by NSF-funded research currently conducted at Oregon State University (OSU) by Assistant Professor and Senior Researcher Andrew Thurber, and Mathematics Professor Malgorzata Peszynska, on the optimization of biofilm growth. This paper explores foundational mathematical models which build toward a real representation of biofilm growth.

Chapter three of this paper presents a simulation of a system of ordinary differential equations representing mass reaction chemical kinetics. Chapter four of this paper presents simulations of one- and two-dimensional diffusion governed by Fick's Second Law of Diffusion. Chapter five of this paper explores a simulation involving two-dimensional diffusion and chemical reactions.

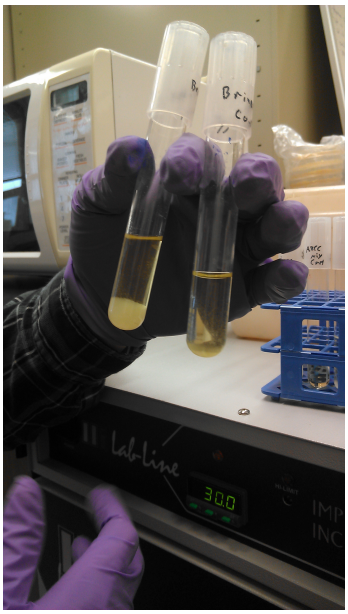

Figure 1.1: Sporosarcina pasteurii grown at Oregon State University under the direction of Senior Researcher Andrew Thurber, referred to in chapter 1.

2. Recent Mathematical Models for Biofilm Growth

Basic characteristics concerning the growth of biofilm with restricted food availability is presented by Eberl and Demaret in paper [5]. Their model is governed by a quasi-linear diffusion equation for biomass density. The model has two causes of degeneracy: diffusion coefficient $D(u)$ dissapears as biomass density nears zero and a singularity appears in $D(u)$ when biomass density approaches its maximum value at 1. They use a nonstandard, finite difference scheme to simulate the model. The key finding in this paper is that food-limited biofilm populations grow in pillars and clusters, whereas non-limited populations grow in flat, homogenous forms. Eberl and Demaret prove this by presenting a simulation scenario in which food nutrients are limited and lysis, the deactivation of biofilm that doesn't have access to food, takes place. In the scenario, biomass grows up until biomass density nears one. The biofilm form then pushes outward, creating pillars and clusters. The model we explore in this paper is a quasi-linear diffusion model, similar to that of Eberl and Demaret. Additionally, we limit nutrient consumption using Monod kinetics and represent biomass density approaching its maximum value at one.

A continuum model for the heterogenous structure of biofilm growth is presented by Eberl, Parker, and Van Loosdrecht in paper [6]. Their model is governed by a quasi-linear diffusion equation for biomass density. The newer development in this paper than earlier biofilm modeling papers is the use of a continuum model for biofilm growth. Earlier papers were exploring probabilistic models with discrete, local conditions. However, among other complications, the probabilistic approach is not physically motivated. Eberl, Parker, and Van Loosdrecht explore one- and three-dimensional structures that are experimentally validated and suggest that biofilm structures have spatial heterogeneities.

3. Mass Reaction

3.1 System of Ordinary Differential Equations (ODEs)

In this chapter, we explore the chemical reactions governed by mass reaction chemical kinetics. Assume we have a container with three interacting chemicals: A, B, and AB. A and B are the reactants that combine to create the product substance AB. At the same time, chemical AB can dissociate, thereby increasing the quantities of chemicals A and B. We denote the concentration level of each chemical by C_A, C_B, and C_{AB}. The reaction that creates chemical AB takes place at a certain rate we denote by k_1. The dissociation reaction of AB takes place at the rate we denote by k_{-1} [7].

$$\mathrm{A} + \mathrm{B} \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} \mathrm{AB} \tag{3.1}$$

Physically, we assume a simple model for these reactions governed by mass reaction chemical kinetics. We assume that the concentration of the reacting substance is the only influence on reaction rate . Thus, the number of possible reactions is proportional to the concentration of each substance [7]. We also assume constant temperature, thus keeping each substance in a consistent phase.

We write these relations defining the rates of reaction and modeling the growth of A, B, and AB as a system of ordinary differential equations (3.1). Note that C_A, C_B, and C_{AB} are functions of time.

$$\begin{aligned} C_A' &= k_{-1}C_{AB} - k_1 C_A C_B \\ C_B' &= k_{-1}C_{AB} - k_1 C_A C_B \\ C_{AB}' &= k_1 C_A C_B - k_{-1}C_{AB} \end{aligned} \tag{3.2}$$

Equation (3.2) defines a system of ordinary differential equations (ODEs) which govern the change of concentration over time for the three reacting chemicals.

Equation (3.2) is a nonlinear, autonomous, first-order system of ODEs. In order for

this system to have a unique solution we need initial conditions. We define these conditions in equation (3.3).

$$\begin{aligned} C_A(t_0) &= C_{Ainit} \\ C_B(t_0) &= C_{Binit} \\ C_{AB}(t_0) &= C_{ABinit} \end{aligned} \tag{3.3}$$

3.2 Solving ODE system

In this section, we explore the numerical approach to understand a solution for equation (3.2).

3.2.1 Numerical Approach

Numerical methods approximate solutions to differential equations. This paper explores a numerical approach utilizing finite difference schemes. Finite difference methods relate to finite difference expressions of the form $f(x+b) - f(x+a)$. These finite difference methods can be understood using basic derivative concepts. Recall the derivative of some function f at a point x is defined by:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.4}$$

Notice the finite difference in the numerator. Suppose h is a fixed, non-zero number. Then we rewrite the derivative to give

$$\frac{f(x+h) - f(x)}{h} = \frac{\Delta_h f(x)}{h} \tag{3.5}$$

This represents the derivative of $f(x)$ for some small h [4].

Given a function of x, $f(x)$, we evaluate equation (3.5) at discrete time values, x_j, to approximate. This is the foundational idea of numerical approximations. x_j represents $x|_j$ where $j = 1, ..., M$. This means that x_j are specific values at which our equation is evaluated. An approximation to the solution is produced by this evaluation.

Given an ordinary differential equation, we choose a numerical method based on stability, convergence, and consistency. We then replace the derivative by the associated finite difference quotient. The resulting combination of differential equation and numerical method can be programmed using MATLAB to generate numerical approximations. This means that the numerator in equation (3.5) may change dependent on the type of numerical method we choose [8].

For a coupled system of differential equations, one must code all equations to run simultaneously in MATLAB. This is because, at any moment, the value of one dependent variable depends on the current value of another dependent variable. The system for chemical kinetics, (3.2), is a coupled system programmed in this simultaneous manner.

3.2.2 MATLAB Implementation

To write the simulation in MATLAB for system (3.2), we create a vector of discrete time points, t_n, at which we approximate the chemical concentration over time. t_n represents $t|_n$ where $n = 1, ..., N$. This means that t_n are specific time values at which the equations in system (3.2) are evaluated. Note that we are evaluating the equations in system (3.2) over time because they are time-dependent ODEs. Simple numerical methods to use are Forward Euler, Backward Euler, 2nd Order Runga-Kutta, and 4th Order Runga Kutta. The disadvantage of Forward Euler is the necessity for a small value between consecutive time points, t_n. We denote this difference as the timestep, h. This small timestep will make the MATLAB function longer to implement than desired. For nonlinear systems, however, like the one explored in this chapter, it is advantageous to use the Forward Euler method because it is simple to program [8].

To implement, write each of the ordinary differential equations in system (3.2) using the forward finite difference quotient in (3.6).

$$D_+u(t) = \frac{u(t+h) - u(t)}{h} \approx u'(t) \tag{3.6}$$

Assume

$$u'(t) = f(u(t), t) \forall t \geq t_0 \tag{3.7}$$

Then rewriting equation (3.6) yields:

$$u(t_{n+1}) \approx u(t_n) + hf(u(t_n), t_n) \tag{3.8}$$

This means that we can approximate the value of our solution, $u(t)$, at any subsequent time value given we know the value and the derivative of $u(t)$ at our current time value. This translates into the following numerical method. Note that U represents the numerical approximation to $u(t)$.

$$U^{n+1} = U^n + hf(U^n, t_n) = U^n + hU'(t_n) \tag{3.9}$$

For chemical A, we create a numerical equation based on equation (3.9), shown in equation (3.10).

$$C_A(t_{n+1}) = C_A(t_n) + hf(C_A(t_n), t_n) \tag{3.10}$$

Here, we generate an equation for $f(C_A(t_n), t_n)$ from system (3.2) since we assume equation (3.7).

$$f(C_A(t_n), t_n) = k_{-1}C_{AB} - k_1 C_A C_B \tag{3.11}$$

We write the full numerical method for system (3.2) in equation (3.12).

$$\begin{aligned} C_A(t_{n+1}) &= C_A(t_n) + h(k_{-1}C_{AB}(t_n) - k_1 C_A(t_n)C_B(t_n)) \\ C_B(t_{n+1}) &= C_B(t_n) + h(k_{-1}C_{AB}(t_n) - k_1 C_A(t_n)C_B(t_n)) \\ C_AB(t_{n+1}) &= C_{AB}(t_n) + h(k_1 C_A(t_n)C_B(t_n) - k_{-1}C_{AB}(t_n)) \end{aligned} \tag{3.12}$$

We evaluate system (3.12) in MATLAB using a vector of timesteps. Let us assume the following initial chemical quantities for chemicals A, B, and AB: $C_{Ainit} = 4$, $C_{Binit} = 3$, and $C_{ABinit} = 7$. Let us assume that it is easier for AB to dissociate into individual chemicals

A and B. Thus, we let $k_{-1} = 1$ and $k_1 = .2$. Evaluating system (3.12) for 10 timesteps with a timestep value $h = 0.1$ generates the plot shown in figure 3.1. The superscript ef identifies that Forward Euler was the method used to compute these values.

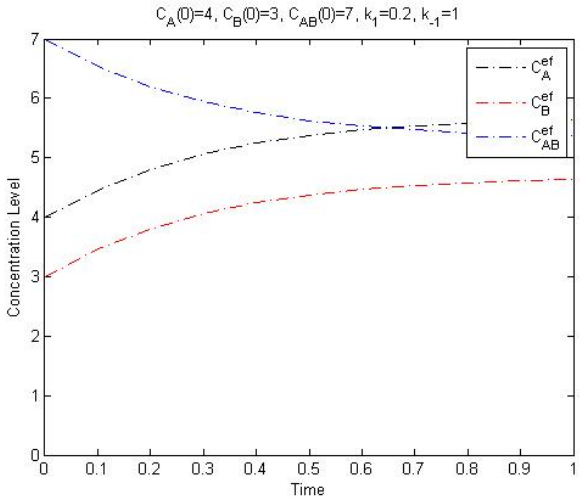

Figure 3.1: Forward Euler sample plot demonstrating chemical concentration over time, referred to in chapter 3.2.2.

3.2.3 Numerical Method Analysis

When utilizing the Forward Euler method, we need to analyze the problem to ensure that we have stability, consistency, and convergence. Stability is governed by the region of absolute stability for a given method. The region of stability for Forward Euler consists of all points inside the unit circle centered at $(-1, 0)$ in the complex plane [8].

Consistency is analyzed by evaluating local truncation error (LTE). The local truncation error for Forward Euler is τ^n within the following equation.

$$\frac{u(t_{n+1}) - u(t_n)}{h} = f(u(t_n), t_n) + \tau^n \tag{3.13}$$

Let us replace $f(u(t_n), t_n)$ by $u'(t_n)$. Then we use Taylor expansions to find $\tau^n = \frac{h}{2}u''(t_n) + O(h^2)$. τ^n is on the order of h but $h\tau^n$ is on the order of h^2 so that at each step you make an h^2 error. By taking $(1/h)$ steps you have an accumulation effect of losing an

h and thus, the full numerical method is on the order of $O(h)$. This tells us that Forward Euler is a first order method [3].

A numerical method is said to converge if the approximation approaches the exact solution as the time step h approaches zero. We know that Forward Euler is convergent [8].

3.2.4 Numerical Method Accuracy

To ensure that the numerical approximation is accurate, we calculate a more accurate approximation using the built-in MATLAB ODE solver, ODE45. We expect the error between these approximations, Forward Euler and ODE45, to be on the order of $O(h)$. We will use five different h values to examine this error.

Shown in figure 3.2 is a plot of both the MATLAB ODE solver and the Forward Euler method. Here, the superscript ef tells us that Forward Euler was used to compute certain values and the superscript $ode45$ tells use that solver ODE45 was used to compute certain values.

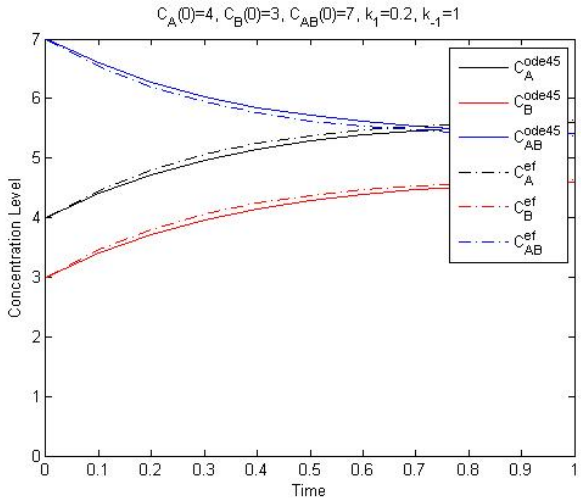

Figure 3.2: Forward Euler method (segmented line) compared to MATLAB ODE45 Solver (solid line), referred to in chapter 3.2.4.

The Forward Euler and ODE45 approximations exhibit similar behavior. However, Forward Euler is not as accurate as ODE45. We compute the error, $E(h)$, using the L_{inf} norm giving the maximum difference between the ODE45 values and the Forward Euler

Error, $E(h)$		
h	$E(h)$	Time (seconds)
0.1	0.2835	0.854111
0.01	0.0257	0.250165
0.001	0.0025	0.903269
0.0001	$2.5449x10^{-4}$	1.350832
0.00001	$2.5839x10^{-5}$	6.280132

Table 3.1: Error for Forward Euler referred to in section 3.2.4.

values. See the errors listed in Table 3.1. Notice that the errors are on the order of the h timestep value. This is in accordance with our knowledge of Forward Euler being on the order of $O(h)$.

4. Diffusion

4.1 Diffusion Partial Differential Equations (PDEs)

This chapter explores diffusion governed by Fick's 2nd Law. It predicts the location of a diffusing substance over time. The following partial differential equation models diffusion in one spatial dimension as an initial boundary value problem (IBVP) [2].

$$u_t - u_{xx} = f(x,t) \tag{4.1}$$

For well-posedness of this IBVP we need to define initial and boundary conditions. This will yield an approximation of a unique solution. For our numerical approach we use Dirichlet boundary conditions where the values at our end points are zero. This means that the spatial boundaries are defined by a fixed value. We define this boundary value using $g(x)$. Our boundary and initial conditions are defined in (4.2).

$$\begin{aligned} u(0,t) &= g(0) = 0 \\ u(M,t) &= g(M) = 0 \\ u(x,0) &= u_{init}(x) \end{aligned} \tag{4.2}$$

The initial conditions will be manually input to our MATLAB simulation to generate a unique approximation. The boundary conditions will be programmed into our MATLAB simulation.

4.2 Numerical Methods for Diffusion

To approximate a solution for the diffusion problem we use a finite difference approach as we did with the mass reaction system. Equation (4.1) is a partial differential equation which is first order for the time variable and second order for the space variable.

4.2.1 Time Discretization

For this first-order derivative, we must choose a finite difference method. Because this is a linear equation, it will be simple and fast to use an implicit method. The implicit Backward Euler method is advantageous because it is characterized by absolute stability for all time step values. We denote our time increment using k and our space increment using h. Assuming

$$u'(t) = f(u(t), t), \ \forall \ t \geq t_0 \tag{4.3}$$

Backward Euler uses the backward difference shown in equation (4.4).

$$D_- u(t) = \frac{u(t) - u(t-k)}{k} \approx u'(t) \tag{4.4}$$

This translates to the numerical method in equation (4.5).

$$\frac{U^{n+1} - U^n}{k} = f(U^{n+1}, t_{n+1}) \tag{4.5}$$

This method is of the order $O(k)$ with a local truncation error of order ck^2 at each step. As well, it is a convergent method.

We rewrite the partial differential equation using Backward Euler and represent our spatial derivative, u_{xx} using matrix AU^{n+1}. Note that U_j^n represents our approximate value of our solution at location j and time h. Here, A represents a matrix.

$$\frac{U_j^{n+1} - U_j^n}{k} = [AU^{n+1}]_j + f(x_j, t_{n+1}) \tag{4.6}$$

Equation (4.6) translates into the numerical method in equation (4.7).

$$U_j^{n+1} = U_j^n + k[AU^{n+1}]_j + kf(x_j, t_{n+1}) \tag{4.7}$$

The subscript j represents the current space grid point corresponding to x_j, in other words, $U|_{x_j} = U_j$ and $U|_{x_j, t_n} = U_j^n$.

4.2.2 Space Discretization

For the second-order space derivative, we must choose a finite difference method to incorporate spatial discretization. To approximate a second-order derivative, we create a numerical method by combining two first-order derivative approximations. We combine the forward and backward finite difference methods to approximate $u''(x)$. This gives us the following approximation in equation (4.8). This serves as a second-order derivative, centered difference.

$$\frac{\frac{u(x+h)-u(x)}{h} - \frac{u(x)-u(x-h)}{h}}{h} = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \approx u''(x) \tag{4.8}$$

This translates to the following numerical method at every x_j, t_n:

$$\frac{1}{h^2}(U_{j+1} - 2U_j + U_{j-1}) \approx u"(x_j) \tag{4.9}$$

4.2.3 Full Discretization

For a full discretization, we substitute the space and time discretizations into the diffusion equation. This results in the following numerical method:

$$\frac{U_j^{n+1} - U_j^n}{k} - \frac{U_j^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}}{h^2} = f(x_j, t_{n+1}). \tag{4.10}$$

4.2.4 MATLAB Implementation

To implement in MATLAB, we solve for U_j^{n+1} and rewrite in matrix form by placing the U values inside a vector over the spatial grid. This form is shown in equation (4.11).

$$\begin{pmatrix} U_0^{n+1} \\ U_1^{n+1} \\ \vdots \\ U_{M-1}^{n+1} \\ U_M^{n+1} \end{pmatrix} + \frac{k}{h^2} \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ & & & & \ddots & & & \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} U_0^{n+1} \\ U_1^{n+1} \\ \vdots \\ U_{M-1}^{n+1} \\ U_M^{n+1} \end{pmatrix} = \begin{pmatrix} g(0) \\ U_1^n \\ \vdots \\ U_{M-1}^n \\ g(M) \end{pmatrix} + k \begin{pmatrix} 0 \\ f(x_0, t_{n+1}) \\ \vdots \\ f(x_{M-1}, t_{n+1}) \\ 0 \end{pmatrix} \tag{4.11}$$

To evaluate, we divide our spatial domain into M equally spaced segments, each the length of our spatial step, h. In order to have M segments we need $M + 1$ grid points, including the end points. For a boundary value problem, it is necessary to define boundary conditions on these end points. We use Dirichlet boundary conditions by which the values at the spatial end points, defined by $g(x)$, stay constant. To implement Dirichlet conditions, the upper and lower rows of the matrix system must be solely dominated by the value of the last timestep. In other words, $U_j^{n+1} = U_j^n$ for $j = 0, M$.

Let us denote the square matrix as A, and we can simply write (4.11) as:

$$(I + \frac{k}{h^2}A)U_j^{n+1} = U_j^n + kf(x_j, t_n) \tag{4.12}$$

Because equation (4.12) is a linear, square system, we can use the MATLAB backslash or inverse operator to solve for U_j^{n+1}. The inverse operator will find the solution using fewer computational steps.

Solving for the U_j^{n+1} vector we obtain the following form:

$$U_j^{n+1} = (I + \frac{k}{h^2}A)^{-1}(U_j^n + kf(x_j, t_n)) \tag{4.13}$$

We program equation (4.13) into MATLAB and iterate for x timesteps. Figure 4.1 displays a sample simulation plot.

In figure 4.1 the green line shows the chemical concentration for the initial time step. The red line shows the chemical concentration for the final time step.

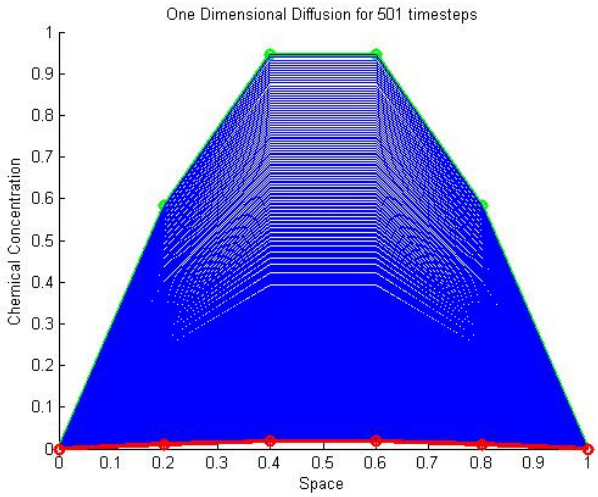

Figure 4.1: One dimensional diffusion with initial chemical concentration of 1 in the center of the space domain referred to in chapter 4.2.4.

4.2.5 Testing Simulation Accuracy

To test the accuracy of this one-dimensional simulation, we assume an exact solution for u, compute the approximation, compute the exact solution, and evaluate the L_{inf} norm error between the two. Let

$$u(x,t) = sin(\pi x)e^{-2t} \tag{4.14}$$

be the exact solution. Then

$$u_t = -2sin(\pi x)e^{-2t} \tag{4.15}$$

and

$$u_{xx} = -\pi^2 sin(\pi x)e^{-2t} \tag{4.16}$$

We see then that

$$f(x,t) = u_t - u_{xx} = (-2 + \pi^2)sin(\pi x)e^{-2t} \tag{4.17}$$

Error, $E(h)$			
h	k	$E(h)$	Time (seconds)
0.2	0.004	$7.5874x10^{-4}$	5.98627
0.1	0.001	$1.9502x10^{-4}$	10.727437
0.066	0.00016	$8.584x10^{-5}$	11.244604
0.04	0.00044	$3.0948x10^{-5}$	54.510372

Table 4.1: Error for one spatial dimension diffusion, referred to in section 4.2.5.

We evaluate this using MATLAB over the same spatial grid and time segment as our previous simulation. Figure 4.2 shows the comparison of our exact solution, using ODE45, with the numerical method approximation, using Backward Euler at the final timestep.

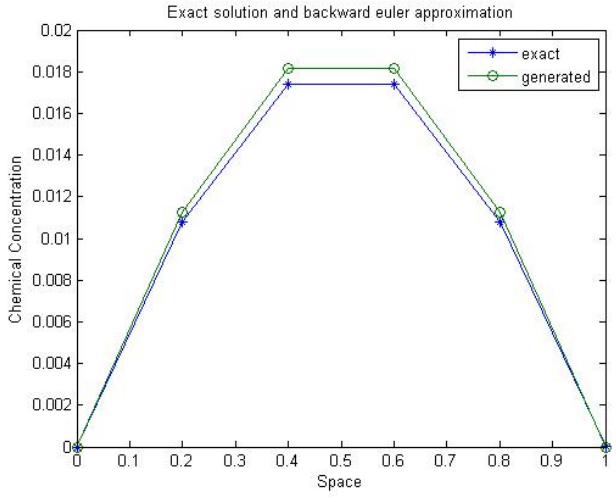

Figure 4.2: Comparison of the exact solution with the Backward Euler generated solution at the final time step, referred to in chapter 4.2.5.

We use the L_{inf} norm to compare error and find that the error is on the order of k^2. For Backward Euler, the expected error is on the order of $O(k)$ and for the second-order derivative method the error is on the order of $O(h^2)$. This is consistent with the error we found in the simulation.

Table 4.1 shows errors dependent on varying spatial step size of h.

4.3 Two Dimensional Diffusion

In this section, we expand the simulation to two spatial dimensions. According to Fick's 2nd Law, the diffusion equation for two dimensions is shown in equation (4.18) with defined initial conditions and boundary conditions in equation (4.19).

$$u_t - \alpha(u_{xx} + u_{yy}) = f(x, y, t) \tag{4.18}$$

$$\begin{aligned} u(0, y, t) = g(0, y) = 0,\ u(M, y, t) = g(M, y) = 0, \\ u(x, 0, t) = h(x, 0) = 0,\ u(x, N, t) = h(x, N) = 0, \\ u(x, y, 0) = u_{init}(x, y) \end{aligned} \tag{4.19}$$

Again, for this numerical approximation we let all of our boundary values equal zero.

4.3.1 Spatial Discretization

To incorporate this second spatial dimension, y, we will apply another second-order derivative method discretization. This gives us the resulting numerical method:

$$\frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{k} - \alpha\left(\frac{U_{i,j+1}^{n+1} - 2U_{i,j}^{n+1} + U_{i,j-1}^{n+1}}{h^2} + \frac{U_{i+1,j}^{n+1} - 2U_{ij}^{n+1} + U_{i-1,j}^{n+1}}{h^2}\right) = f(x_i, y_j, t_n) \tag{4.20}$$

Note that U_j^n approximates $u(x_j, t_n)\ \forall\ x_j, t_n$.

We simplify this method as follows in equation (4.21).

$$U_{i,j}^{n+1} = \frac{k\alpha}{h^2}(U_{i-1,j}^{n+1} + U_{i+1,j}^{n+1} - 4U_{i,j}^{n+1} + U_{i,j-1}^{n+1} + U_{i,j+1}^{n+1}) + U_{i,j}^{n} + kf(x_i, y_j, t_n) \tag{4.21}$$

Combining our $U_{i,j}^{n+1}$ terms from both spatial discretizations in x and y gives us the resulting 4 coefficient in equation (4.21).

4.3.2 MATLAB Implementation

Rewriting this in matrix form requires the use of square matrix A which governs the two-dimensional spatial discretization.

First, let us examine the unpacking of the matrix u, the matrix containing the chemical concentration values. Let us use the term *umatrix* for this matrix form of u and *uvector* for the vector form of u. Below is the matrix *umatrix* which shows the x and y values corresponding to each matrix element on our rectangular grid. This matrix is an n by m matrix. Here, n is the maximum value in the y-space direction and m is the maximum value in the x-space direction. To explore an example, let $m = 24$ and $n = 13$.

$$umatrix = \begin{pmatrix} (1,1) & (1,2) & (1,3) & \cdots & (1,24) \\ (2,1) & (2,2) & (2,3) & \cdots & (2,24) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (13,1) & (13,2) & (13,3) & \cdots & (13,24) \end{pmatrix} \tag{4.22}$$

In order to implement the numerical method, we need to generate a u vector from the values in *umatrix*. To do this we create an algorithm called *myind*. *myind* lists the values in *umatrix* in vector form moving from left to right and top to bottom. Thus, the following vector shows how *umatrix* values are shifted into the *uvector*.

$$uvector = \begin{pmatrix} (1,1) \\ (1,2) \\ (1,3) \\ \vdots \\ (1,13) \\ (2,1) \\ (2,2) \\ (2,3) \\ \vdots \\ (2,13) \\ (3,1) \\ (2,2) \\ (3,3) \\ \vdots \\ (24,13) \end{pmatrix} \tag{4.23}$$

Note that the size of *umatrix* is 13 by 24 and the size of *uvector* is 312 by 1.

We simplify the method, allow A to represent the coefficients from our spatial discretization, and let A absorb $\frac{k\alpha}{h^2}$. This simplification is shown equation (4.24).

$$(I + A)U^{n+1} = kf^{n+1} + U^n \tag{4.24}$$

In equation (4.24), U is the numerical approximation to our solution $u(x, y, t)$.

Now, let us explore the form of matrix A, shown below. To generate the understanding let us think about two smaller domains. First, we'll show a 3 by 3 domain and second a 4 by 4 domain. These visualizations will give us a foundation to understand the form of matrix A for a 13 by 24 rectangular grid domain.

$$A_{3x3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{k\alpha}{h^2} & 0 & -\frac{k\alpha}{h^2} & 4\frac{k\alpha}{h^2} & -\frac{k\alpha}{h^2} & 0 & -\frac{k\alpha}{h^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.25}$$

In matrix A_{3x3} in equation (4.25), row 5 is the most detailed row. This is because row five represents the only internal grid point, namely point $(2, 2)$. We leave the other rows without detail because we will be using Dirichlet boundary conditions. The diagonal ones represent that each of the boundary points retains the same valued defined by $g(x)$ for every time step. This means that the values on the boundaries do not change over time. Now, we explore the 4 by 4 domain case.

$$A_{4x4} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 4\frac{k\alpha}{h^2} & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 4\frac{k\alpha}{h^2} & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 4\frac{k\alpha}{h^2} & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 4\frac{k\alpha}{h^2} & -\frac{k\alpha}{h^2} & 0 & 0 & -\frac{k\alpha}{h^2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{4.26}$$

Again, notice that the detailed rows correspond to the internal grid points of the domain. For a rectangular domain, we simply align equation detail with the internal grid points of the domain. Imagine this layout expanded to a 24 by 13 grid which we explore in the following example. The only addition needed for a non-square domain is that we do not only have h terms in our denominators. We will have h_x and h_y terms to differentiate different space increments in each spatial dimension.

In this example, let α, the diffusion coefficient, equal 1. We compute the Backward Euler approximation for 100 time steps using the time step increment $k = .01$. Figure 4.3 represents the displayed approximation at one time step.

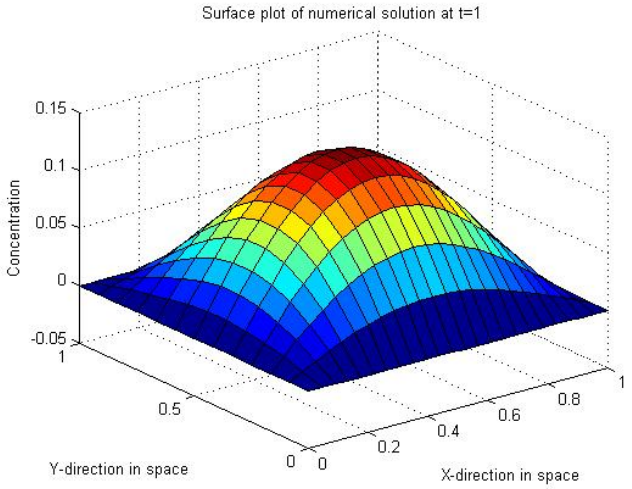

Figure 4.3: Backward Euler approximation in two spatial dimensions over a 24 by 13 rectangular domain, referred to in chapter 4.3.2.

4.3.3 Numerical Method Analysis

Now, we analyze the numerical method. In order to to do this, we choose a sample exact solution, $u(x, y, t)$. Let

$$u(x, y, t) = sin(\pi x)sin(\pi y)e^{-2t} \tag{4.27}$$

Then the forcing function is

$$f(x, y, t) = u_t - \alpha(u_{xx} + u_{yy}) = sin(\pi x)sin(\pi y)e^{-2t}(-2 + 2\alpha\pi^2) \tag{4.28}$$

We implement and plot this exact solution over the spatial and time grids used in figure 4.3. See the results compared in figure 4.4.

4.3.4 Testing Simulation Accuracy

We generate the L_{inf} norm error between the sample exact solution and the numerical approximation. The error is on the order of $O(k) + O(h^2)$. This is consistent with the error we expect for Backward Euler and centered difference. Table 4.2 shows the error values

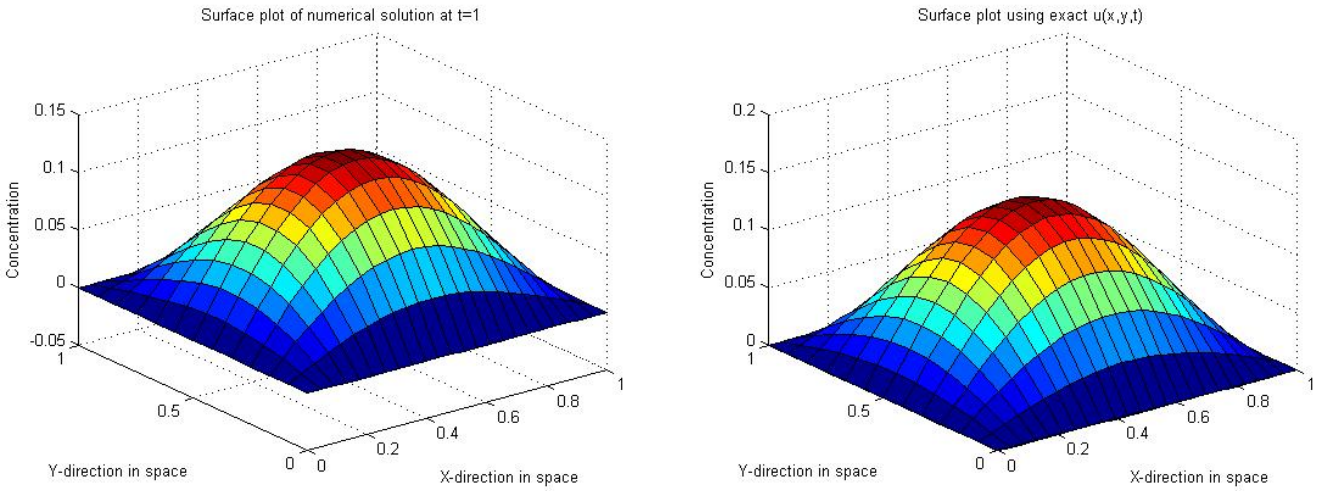

Figure 4.4: Comparison of the proxy exact solution (right figure) with the Backward Euler generated solution (left figure) at the final time step referred to in chapter 4.3.3.

dependent on time and space steps.

Error, $E(h)$					
hy	hx	k	$E(h)$	Time (seconds)	Quality of Behavior
0.5	0.4166	0.01	N/A	N/A	N/A
0.25	0.4166	0.01	0.0863	51.676536	Poor
0.625	0.4166	0.01	0.0175	52.3366631	Good
0.03125	0.4166	0.01	0.0147	52.522455	Good
0.0769	0.5	0.01	N/A	N/A	N/A
0.0769	0.25	0.01	0.0855	53.605038	Poor
0.0769	0.625	0.01	0.0126	53.314103	Good
0.0769	0.03125	0.01	0.0081	55.685118	Good
0.0769	0.4166	0.1	0.1350	52.437321	Good
0.0769	0.4166	0.01	0.0154	52.353048	Good
0.0769	0.4166	0.001	0.6602	52.482394	Good
0.0769	0.4166	0.0001	0.8487	52.219641	Good

Table 4.2: Error for two-dimensional spatial simulation referred to in section 4.3.4.

5. Reaction-Diffusion

5.1 Two Species System

This chapter explores nutrient diffusion and biofilm growth. First, we form the partial differential equations that govern nutrient and biofilm in two spatial dimensions. Then we run simulations of the resulting numerical method. This implementation is developed using the IMEX scheme numerical method. This means we discretize our diffusion terms implicitly and the reaction terms explicitly.

5.1.1 Differential Equations

In this situation, the nutrient will diffuse in two spatial dimensions. Thus, we start with the nutrient, $u(x, y, t)$, governed by (4.18). We also know the nutrient will be consumed by the biofilm. So, we now add a $-k_2bu$ term, where $b(x, y, t)$ represents biofilm concentration. By Monod kinetics, we divide the $-k_2bu$ term by $1 + u(x, y, t)$, which represents the critical point where biofilm stops growth after consuming all available nutrients. The results are shown in equation (5.1).

$$u_t = \alpha(u_{xx} + u_{yy}) - \frac{k_2bu}{1+u} \tag{5.1}$$

For the biofilm differential equation, we only need to represent growth by the consumption of nutrient. We represent this in equation (5.2).

$$b_t = \frac{k_3bu}{1+u} \tag{5.2}$$

The full system is now shown in equation (5.3).

$$\begin{aligned} u_t &= a(u_{xx} + u_{yy}) - \tfrac{k_2bu}{1+u} \\ b_t &= \tfrac{k_3bu}{1+u} \end{aligned} \tag{5.3}$$

System (5.3) is a nonlinear, coupled system of partial differential equations.

5.1.2 Numerical Method

We use the IMEX numerical method by which diffusion is treated implicitly and reaction is treated explicitly. We choose to discretize in this way due to the level of efficiency leveraged by the IMEX method. We use the second-order derivative, centered difference approximation shown in (4.8) for our space discretization and we use Backward Euler for our time discretization. We let the column vector U represent the numerical approximation for $u(x, y, t)$ and column vector B represent the numerical approximation for $b(x, y, t)$. We are using the positive definite matrix A modeled after the sample matrices shown in (4.3.2) with $\frac{}{h^2}$ absorbed into matrix A. We also define a column vector P^n whose entries are given by term (5.4) arranged in a manner corresponding to the vector reorientation sampled by the *uvector* shown in equation (4.23). Column vectors B and U are also arranged corresponding to this reorientation so that each point on the rectangular grid is represented in our columns.

$$\frac{k_2 B_{i,j}^n U_{i,j}^n}{1 + U_{i,j}^n} + U_{i,j}^n, \ \forall \ \mathrm{i, j}. \tag{5.4}$$

Given B, U, and P^n the numerical method used to approximate our nutrient is shown in equation (5.5).

$$U^{n+1} = (I + A)^{-1} - kP^n, \ \forall \ \mathrm{i, j}. \tag{5.5}$$

Using Forward Euler method for biofilm, we obtain the following biofilm numerical method in equation (5.6).

$$B_{i,j}^{n+1} = \frac{k k_3 B_{i,j}^n U_{i,j}^n}{1 + U_{i,j}^n} + B_{i,j}^n, \ \forall \ \mathrm{i, j}. \tag{5.6}$$

5.1.3 MATLAB Implementation

In order to test these numerical methods in MATLAB, we create random biofilm regions in the domain to serve as our initial conditions. Now, the implementation outputs two figures to show nutrient diffusion as well as biofilm growth. Here is a sample when $k = 0.0100$, $h_x = 0.0833$, $h_y = 0.0435$, $k_2 = 3$, $k_3 = 9$.

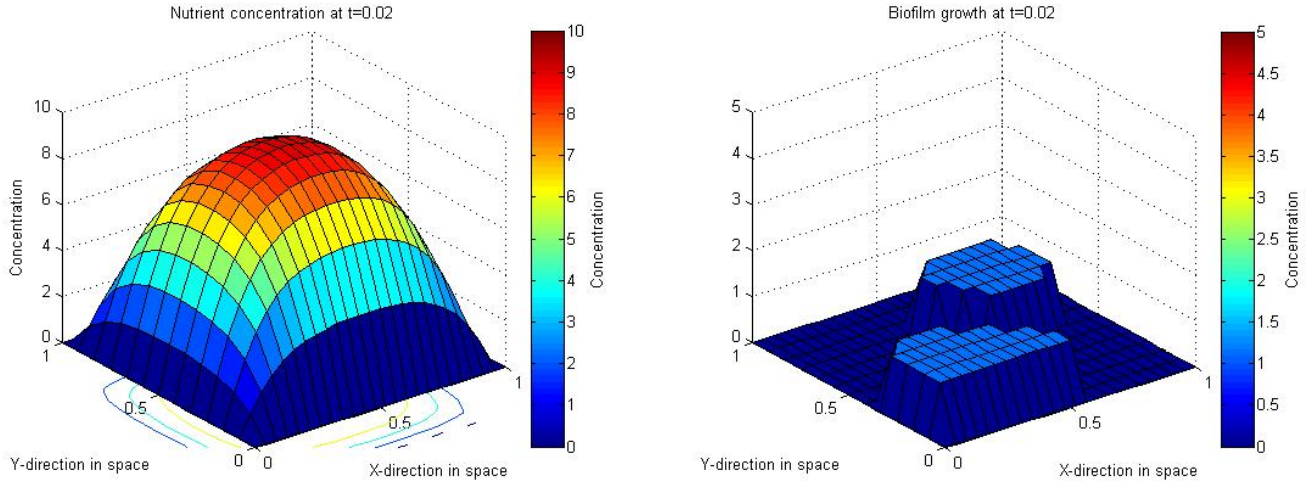

Figure 5.1: Time: 0.02. Backward Euler approximation of the nutrient, u (left figure). Forward Euler approximation of the biofilm, b (right figure) referred to in chapter 5.1.3.

Due to the complicated nature of the two-dimensional reaction-diffusion problem we cannot easily understand the accuracy of our simulation. While we cannot fully trust our implementation, we expect reasonable accuracy given that this implementation is built from the previous simulations described in this paper.

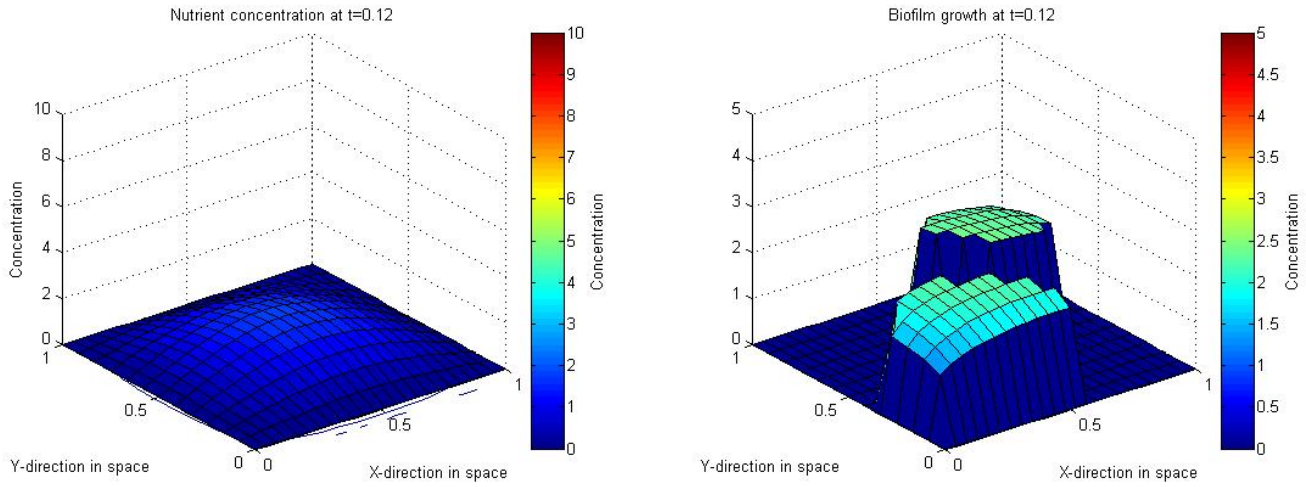

Figure 5.2: Time: 0.12. Backward Euler approximation of the nutrient, u (left figure). Forward Euler approximation of the biofilm, b (right figure) referred to in chapter 5.1.3.

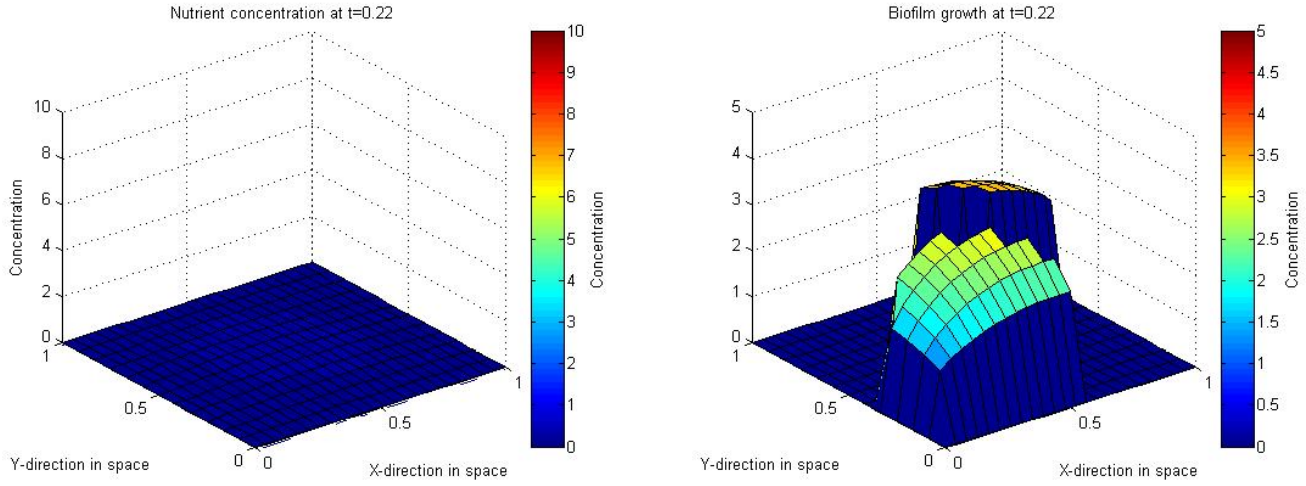

Figure 5.3: Time: 0.22. Backward Euler approximation of the nutrient, u (left figure). Forward Euler approximation of the biofilm, b (right figure) referred to in chapter 5.1.3.

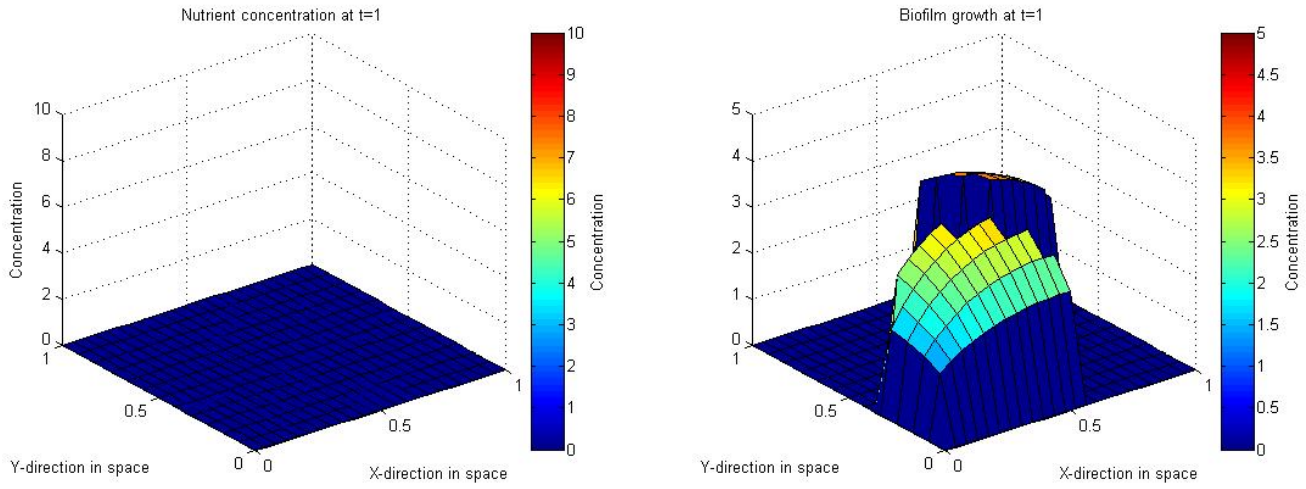

Figure 5.4: Time: 1.00. Backward Euler approximation of the nutrient, u (left figure). Forward Euler approximation of the biofilm, b (right figure) referred to in chapter 5.1.3.

6. Conclusion

A foundational spatio-temporal model is developed yielding simulations to predict biofilm growth. To aide CO_2 sequestration research, this model establishes the beginning components of a more comprehensive model. The development of this model began with the exploration of modeling chemical reactions dependent on time. Next, modeling of diffusion was developed in one spatial dimension. This was followed by the development of a two-dimensional spatial model of biofilm growth. Finally, this paper implemented and tested a numerical IMEX scheme method for a coupled system of two species in two-dimensions of reaction-diffusion type.

Some further extensions that this paper motivates are the addition of flow processes to this model. Additionally, one could add another species of competing biofilm, three spatial dimensions, and a singularity for biofilm density at a value of 1. A necessary extension is accuracy testing for the coupled IMEX system.

In summary, through working on this project I studied approximations to ODEs and PDEs and implemented and tested a numerical method (IMEX) for a coupled system of two species in two dimensions of reaction-diffusion type. I also learned about mathematical modeling of biofilm growth and tested numerical method accuracy of schemes on simples examples. Additionally, I gained technical skills using MATLAB, LaTex, and Beamer.

Appendix A

Forward Euler, Chemical Reactions

```
function EF_conc(N, Ca_0, Cb_0, Cab_0, k_1, k_inv1)
k = 1/N;
tsteps = zeros(N+1, 1);
Cavalues = zeros(N+1, 1);
Cbvalues = zeros(N+1, 1);
Cabvalues = zeros(N+1, 1);
Cavalues(1) = Ca_0;
Cbvalues(1) = Cb_0;
Cabvalues(1) = Cab_0;
tsteps(1) = 0;
for n = 1:N
    tsteps(n+1) = tsteps(n) + k;
    Cavalues(n+1) = Cavalues(n) + k.*((k_inv1).*Cabvalues(n) - (k_1).*...
        Cavalues(n).*Cbvalues(n));
    Cbvalues(n+1) = Cbvalues(n) + k.*((k_inv1).*Cabvalues(n) - (k_1).*...
        Cavalues(n).*Cbvalues(n));
    Cabvalues(n+1) = Cabvalues(n) + k.*((k_1).*Cavalues(n).*Cbvalues(n) - (...
        k_inv1).*Cabvalues(n));

end
plot(tsteps, Cavalues, 'k-.', tsteps, Cbvalues, 'r-.', tsteps, Cabvalues, 'b-...
    .')
axis([0, 1, 0, max([Ca_0, Cb_0, Cab_0])])
legend('C_A^{ef}', 'C_B^{ef}', 'C_{AB}^{ef}')
title(['C_A(0)=' num2str(Ca_0), ', ' 'C_B(0)=' num2str(Cb_0), ', '  'C_{AB...
    }(0)=' num2str(Cab_0), ', ' 'k_1=' num2str(k_1), ', ' 'k_{-1}=' num2str(...
    k_inv1)])
xlabel('Time')
ylabel('Concentration Level')
z = [Cavalues(:,1) Cbvalues(:,1) Cabvalues(:,1)]
size(z)
```

Appendix B

ODE45 Solver, Chemical Reactions

```
function mass_reacf(N, Ca_0, Cb_0, Cab_0, k_1, k_inv1)
k = 1/N;
tspan = (0:k:1)'
A = Ca_0
B = Cb_0
C = Cab_0
yzero = [A B C]

[t,y] = ode45(@mass_reacf, tspan, yzero)
size(y),
y,
clf;
plot(t, y(:,1),'k',t,y(:,2),'r',t,y(:,3),'b');
legend('C_A^{ode45}','C_B^{ode45}','C_{AB}^{ode45}');
axis([0, 1, 0, max(yzero)])
title(['C_A(0)=' num2str(Ca_0) 'C_B(0)=' num2str(Cb_0) 'C_{AB}(0)=' num2str(...
    Cab_0) 'k_1=' num2str(k_1) 'k_{-1}=' num2str(k_inv1)])
xlabel('Time')
ylabel('Concentration Level')

%---------------Nested functions-----------------
function yprime = mass_reacf(t,y)

    yprime = zeros(3,1);
    yprime(1)=(k_inv1).*y(3) - (k_1).*y(1).*y(2);
    yprime(2)=(k_inv1).*y(3) - (k_1).*y(1).*y(2);
    yprime(3)=(k_1).*y(1).*y(2) - (k_inv1).*y(3);

end
end
```

Appendix C

Backward Euler, One Dimensional Diffusion

```
function backeulwsn(M, x_f, T)

h = x_f./M; % I think this is arbitratily chosen to be a small space step.
k = (h.^2)/10; %This time step was chosen for stability reasons.

%The set up of the equation with source term f(x,t) = 0 is the form:
%
%0 = (u^{n+1} - u^{n})/k + (1/h^2).*A*u^{n+1}
%
%And then the form:
%
%u^{n+1} + (k/h^2).*A*u^{n+1} = u^{n} where A is chosen in accordance with
%second order, centered difference approximation (1/h^2)(-u_{j-1} + 2u_{j}
%-u_{j+1}) each evaluated at n. In order ot find u^{n+1},
%our next step data values, we must use the backslash operator in the form:
%
%(I + (k/h^2).*A)*u^{n+1} = u^{n}
%
%And then the form:
%[Recall Backslash Operator: Ax = B goes to x = A\B].
%
%u^{n+1} = (I + (k/h^2).*A)\u^{n}.
%
%We modify the above equation with a source term, f(x,t) creating this form:
%
%f(x,t) = (u^{n+1} - u^{n})/k + (1/h^2).*A*u^{n+1}
%
%And then the form:
%
%k.*f(x,t) = (I + (k/h^2).*A)*u^{n+1} - u^{n}
%
%And then the form:
%
%(I + (k/h^2).*A)*u^{n+1} = k.*f(x,t) + u^{n}
%
%Solving for u^{n+1} using the backslash operator gives:
%
%u^{n+1} = (I + (k/h^2).*A)\(k.*f(x,t) + u^{n})
```

```
%
%%For simplification in this code, we will use the following notation:
%A = matrix representing centered difference w/o h^2
%B = I + ((k/h^2).*A)
%F = k.*f(x,t)
%K = k.*f(x,t) + u^{n} = F + u^{n}

%Note: We have M space segements of length h. Thus we
%record data for M+1 points that
%seperate our M space segments.

%First, let's create our space step vector, u vector, matrix A, and matrix B.

xsteps = (0:h:x_f)'; %This is our vector with space step points.
uvalues = sin(pi.*xsteps); %This vector gives our initial condition,
%i.e. our numerical data at t=0.
uvalues(1,1) = 0;
uvalues(M+1, 1) = 0;


A = zeros(M+1);
for j = 2:M
    A(j,j) = 2; A(j,j-1) = -1; A(j,j+1) = -1;
end

%The above for-loop generates A with 2 on the diagonal, -1 on the
%semi-diagonals and the (1,1) and (M+1, M+1) elements equal to zero in
%order to generate our boundary conditions.

C = (k./(h^2)).*A;
B = eye(M+1) + C;

%Second, let's create our vectors F and K.

%We define f(x,t) = u_{t} - u_{xx} where u(x,t) = sin(pi.*x).*exp(-2.*t).
%Thus, f(x,t) = (-2 + pi^2).*sin(pi.*x).*exp(-2.*t).
%So, our vector F will depend on both time and space. We have a vector,
%xsteps, that will determine our values of x. To determine our changing
%values of t let us use a time for-loop. We need this loop to take on
%non-integer values. Thus we create list v which holds all of the time values
%over which we wish to evaluate.
%
%Recall:
%F = k.*f(x,t)
%K = k.*f(x,t) + u^{n} = F + u^{n}
%
%To be used in:
%u^{n+1} = (I + (k/h^2).*A)\(k.*f(x,t) + u^{n})
%
%Using our f(x,t) we then create our vector F and K. We execute the
%backslash operator in order to find u^{n+1} within our for-loop.

v = (0:k:T);
for n = v
    f = (-2+(pi^2)).*sin(pi.*xsteps).*exp(-2.*n);
    f(1, 1) = 0;
    f(M+1, 1) = 0;
    F = k.*f;
    K = F + uvalues;
```

```
    uvalues = B\K;
    hold on
    plot(xsteps, uvalues)
    if n == 0
        plot(xsteps, uvalues, 'go-', 'LineWidth', 2.5)
    else
        plot(xsteps, uvalues, 'b')
    end
    if n == T
        plot(xsteps, uvalues, 'ro-', 'LineWidth', 2.5)
    else
        plot(xsteps, uvalues, 'b')
    end
end
rr = size(v)
title(['One dimensional diffusion for ', num2str(rr(2)), ' timesteps'])
ylabel('Chemical Concentration')
xlabel('Space')


p
%Now, let's plot the exact value dependent on the following equation:

%u(x,t) = sin(pi.*x).*exp(-2.*t).

vvalues = sin(pi.*xsteps).*exp(-2.*T);

figure
plot(xsteps, vvalues, '*-', xsteps, uvalues, 'o-')
legend('exact', 'generated')
title(['Exact solution and backward euler approximation'])
ylabel('Chemical Concentration')
xlabel('Space')


%We want to include the dimensions of our vector v, vector K, and matrix B
%to help with troubleshooting.

%x = size(v)
%y = size(B)
%z = size(K)

E = norm(vvalues - uvalues, inf)
```

Appendix D

Backward Euler, Two Dimensional Diffusion

```
function sample2becd(m, n, k, a, NT)
tic()
%Create our spacesteps, specify our space steps for x and y

x = linspace(0, 1, m);
y = linspace(0, 1, n);
hx = 1/(m-1);
hy = 1/(n-1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create x(j) and y(l) functions

    function [q] = xx(j)
        q = j*hx;
    end

    function [r] = yy(l)
        r = l*hy;
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Create the packing function
    function [p] = myind(j,l,n)
        p = n.*(j-1) + l;
    end

A = sparse(m*n,m*n);

%Create A, the matrix that corresponds to centered difference in
%x and y
for j = 2:m-1
    for l = 2:n-1
        A(myind(j,l,n), myind(j+1,l,n)) = - a./(hx)^2;
        A(myind(j,l,n), myind(j-1,l,n)) = - a./(hx)^2;
        A(myind(j,l,n), myind(j,l+1,n)) = - a./(hy)^2;
        A(myind(j,l,n), myind(j,l-1,n)) = 4- a./(hy)^2;
        A(myind(j,l,n), myind(j,l,n)) = 2.*((1./hx)^2 + (1./hy)^2);
```

```
39      end
40  end
41
42  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
43
44  %Create B, the matrix we use in our backslash operation
45  I = eye(m.*n);
46  B = I + k.*A;
47
48  %Check the eigenvalues for stepsize/stability information
49  %eig(A);
50  %eig(B);
51
52  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53
54  %Create our values at t=0 and boundary conditions (zeros on boundary)
55  %This uses u(x,y,t) = sin(pix)sin(piy)e^-2t
56  uold = zeros(m.*n, 1);
57  for j = 2:m-1
58      for l = 2:n-1
59          uold(myind(j,l,n)) = sin(xx(j).*pi).*sin(yy(l).*pi);
60      end
61  end
62
63  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
64
65  %Create our XX and YY meshgrids for our surface plot
66  [XX,YY] = meshgrid(x,y);
67  P = [XX(:), YY(:)];
68
69  %%%%%%%%%%%%%%%%%%%
70  %Unpack our values from uold into matrix form in order to plot using
71  %surf/mesh
72  for v = 1:m.*n
73      for l = 1:n
74          g = (v-l)./n;
75          if mod(g,1) == 0
76              j = ((v-l)./n) + 1;
77              uoldm(l,j) = uold(v);
78          end
79      end
80  end
81  %%%%%%%%%%%%%%%%%%%
82
83  %Plot our XX,YY,uoldm matrices to physically see our data
84  %surfc(XX, YY, uoldm);
85  %xlabel('X'); ylabel('Y'); zlabel('Z'); figure
86
87  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
88
89  %Now, we want to iterate over our timesteps w
90  %First, generate f using f(x,y,t) = u_t -a(u_xx + u_yy) =
91  %sin(pix)sin(piy)e^(-2t)(-2+2api^2)
92  figure
93
94  for w = 1:NT
95      tw = k.*w;
96      f = zeros(m.*n, 1);
97      for j = 2:m-1
```

```
98          for l = 2:n-1
99              f(myind(j,l,n)) = sin(xx(j).*pi).*sin(yy(l).*pi)*(exp(-2.*(tw))....
                    *(-2 + 2.*a.*pi.^2);
100         end
101     end
102     rhs = k.*f + uold; %This is our right hand side
103     unew = B\rhs; %Now, we solve the linear system.
104
105     %%%%%%%%%%%%%%%%%%%%
106     %Unpack our unew vector into matrix form to plot using surf/mesh
107     for v = 1:m.*n
108         for l = 1:n
109             g = (v-l)./n;
110             if mod(g,1) == 0
111                 j = ((v-l)./n) + 1;
112                 unewm(l,j) = unew(v);
113             end
114         end
115     end
116     %%%%%%%%%%%%%%%%%%%%
117
118     surf(XX, YY, unewm) %Plot at time w
119     xlabel('X-direction in space'); ylabel('Y-direction in space');
120     zlabel('Concentration');
121     title(sprintf('Surface plot of numerical solution at t=%g',tw))
122     pause(.5)
123     uold = unew; %Prepare the data for the next iteration
124 end
125 size(unewm)
126 toc()
127 end
```

Appendix E

Two Species

```
function sample2becdREACTION(m, n, a, NT)

%Create our spacesteps, specify our space steps for x and y

k = 1/NT;
x = linspace(0, 1, m);
y = linspace(0, 1, n);
hx = 1/(m-1);
hy = 1/(n-1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create x(j) and y(l) functions

    function [q] = xx(j)
        q = j*hx;
    end

    function [r] = yy(l)
        r = l*hy;
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Create the packing function
    function [p] = myind(j,l,n)
        p = n.*(j-1) + l;
    end

A = sparse(m*n,m*n);

%Create A, the matrix that corresponds to centered difference in
%x and y
for j = 2:m-1
    for l = 2:n-1
        A(myind(j,l,n), myind(j+1,l,n)) = - a./(hx)^2;
        A(myind(j,l,n), myind(j-1,l,n)) = - a./(hx)^2;
        A(myind(j,l,n), myind(j,l+1,n)) = - a./(hy)^2;
        A(myind(j,l,n), myind(j,l-1,n)) = - a./(hy)^2;
        A(myind(j,l,n), myind(j,l,n)) = 2.*((1./hx)^2 + (1./hy)^2);
    end
end
```

```
42
43 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
44
45 %Create B, the matrix we use in our backslash operation
46 I = eye(m.*n);
47 B = I + k.*A;
48
49 %Check the eigenvalues for stepsize/stability information
50 %eig(A);
51 %eig(B);
52
53 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
54
55 %Create our values at t=0 and boundary conditions (zeros on boundary)
56 %This uses u(x,y,t) = sin(pix)sin(piy)e^-2t
57 uold = zeros(m.*n, 1);
58 for j = 2:m-1
59     for l = 2:n-1
60         uold(myind(j,l,n)) = 10; %55 sin(xx(j).*pi).*sin(yy(l).*pi);
61     end
62 end
63
64 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
65
66 %Create our XX and YY meshgrids for our surface plot
67 [XX,YY] = meshgrid(x,y);
68 P = [XX(:), YY(:)];
69
70 %%%% setup bold= initial condition for biofilm
71 bold = 0*uold;
72 rand('seed',0);
73 %% set up random centers and radii of blobs
74 k1 = 3; k2 = 9;
75
76 x1 = rand; y1 = rand;
77 x2 = rand; y2 = rand;
78 r1 = rand; r1 = 0.1 + r1*0.3;
79 r2 = rand; r2 = 0.1 + r2*0.3;
80 for j = 2:m-1
81     for l = 2:n-1
82         blob = 0;
83         if norm([x1,y1]-[x(j),y(l)]) <r1, blob = 1;end
84         if norm([x2,y2]-[x(j),y(l)]) <r2, blob = 1;end
85         bold(myind(j,l,n)) = blob;
86     end
87 end
88
89 %%%%%%%%%%%%%%%%%%
90 %Unpack our values from uold into matrix form in order to plot using
91 %surf/mesh
92 for v = 1:m.*n
93     for l = 1:n
94         g = (v-l)./n;
95         if mod(g,1) == 0
96             j = ((v-l)./n) + 1;
97             uoldm(l,j) = uold(v);
98         end
99     end
100 end
```

```
101
102 %%%%%%%%%%%%%%%%%%
103 %Unpack our values from uold into matrix form in order to plot using
104 %surf/mesh
105 for v = 1:m.*n
106     for l = 1:n
107         g = (v-l)./n;
108         if mod(g,1) == 0
109             j = ((v-l)./n) + 1;
110             boldm(l,j) = bold(v);
111         end
112     end
113 end
114
115 %%%%%%%%%%%%%%%%%%%
116 %Plot our XX,YY,uoldm matrices to physically see our data
117 %surfc(XX, YY, uoldm);
118 %xlabel('X'); ylabel('Y'); zlabel('Z'); figure
119
120 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
121
122 %Now, we want to iterate over our timesteps w
123 %First, generate f using f(x,y,t) = u_t -a(u_xx + u_yy) =
124 %sin(pix)sin(piy)e^(-2t)(-2+2api^2)
125 figure
126
127 for w = 1:NT
128     tw = k.*w;
129     f = zeros(m.*n, 1); g = zeros(m.*n, 1);
130     for j = 2:m-1
131         for l = 2:n-1
132         %%%    f(myind(j,l,n)) = sin(xx(j).*pi).*sin(yy(l).*pi)*(exp(-2.*(tw)...
            )).*(-2 + 2.*a.*pi.^2);
133             f(myind(j,l,n)) = (-k1*bold(myind(j,l,n))*uold(myind(j,l,n))).*...
                /(1+uold(myind(j,l,n)));
134             g(myind(j,l,n)) = (k2*bold(myind(j,l,n))*uold(myind(j,l,n))./(1+...
                uold(myind(j,l,n)));
135         end
136     end
137     rhs = k.*f + uold; %This is our right hand side
138     unew = B\rhs; %Now, we complete our backslash operation
139
140     bnew = bold + k*g;
141
142     %%%%%%%%%%%%%%%%%%%%%
143     %Unpack our unew vector into matrix form to plot using surf/mesh
144     for v = 1:m.*n
145         for l = 1:n
146             g = (v-l)./n;
147             if mod(g,1) == 0
148                 j = ((v-l)./n) + 1;
149                 unewm(l,j) = unew(v);
150             end
151         end
152     end
153     %%%%%%%%%%%%%%%%%%%%%
154     %Unpack our unew vector into matrix form to plot using surf/mesh
155     for v = 1:m.*n
156         for l = 1:n
```

```
157             g = (v-1)./n;
158             if mod(g,1) == 0
159                 j = ((v-1)./n) + 1;
160                 bnewm(1,j) = bnew(v);
161             end
162         end
163     end
164     %%%%%%%%%%%%%%%%%%%
165
166     figure(1);surfc(XX, YY, unewm) %Plot at time w
167     axis([0, 1, 0, 1, 0, 10]);
168     caxis([0, 10]);
169     xlabel('X-direction in space'); ylabel('Y-direction in space'); zlabel('...
            Concentration');
170     title(sprintf('Nutrient concentration at t=%g',tw))
171     aa = colorbar; %caxis([0 10]);
172     ylabel(aa, 'Concentration');
173     figure(2);surfc(XX, YY, bnewm) %Plot at time w
174     axis([0, 1, 0, 1, 0, 5]);
175     caxis([0, 5]);
176     xlabel('X-direction in space'); ylabel('Y-direction in space'); %zlabel('...
            Z');
177     title(sprintf('Biofilm growth at t=%g',tw))
178     bb = colorbar; %caxis([0 1]);
179     ylabel(bb, 'Concentration');
180     pause(.5)
181     uold = unew; %Prepare the data for the next iteration
182     bold = bnew;
183 end
184
185 end
```

Bibliography

[1] Randy Hiebert Robin Gerlach Lee H. Spangler Alfred B. Cunningham Andrew C. Mitchell, Adrienne J. Phillips, *Biofilm enhanced geologic sequestration of supercritical co_2*, International Journal of Greenhouse Gas Control **3** (2009), 90–99.

[2] P. W. Atkins, *Physical chemistry*, Oxford University Press, Inc., 1986, Third Edition.

[3] Kendall Atkinson and Weimin Han, *Elementary numerical analysis*, John Wiley and Sons, Inc., 2004, Third Edition.

[4] William G. McCallums et al. Deborah Hughes-Hallet, Andrew G. Gleason, *Calculus, single and multivariable*, John Wiley and Sons, Inc., 2005, Fourth Edition.

[5] Laurent Demaret Hermann J. Eberl, *A finite difference scheme for a degenerated diffusion equation arising in microbial ecology*, Electronic Journal of Differential Equations **15** (2007), 77–94.

[6] Mark C.M. Van Loosdrecht Hermann J. Eberl, David F. Parker, *A new deterministic spatio-temporal continuum model for biofilm development*, Journal of Theoretical Medicine **3** (2001), 161–175.

[7] Albert L. Lehninger, *Biochemistry*, Worth Publishers, Inc., New York, 1975, Second Edition.

[8] Randall J. LeVeque, *Finite difference methods for ordinary and partial differential equations*, Society for Industrial and Applied Mathematics, Philadelphia, 2007.